ROB-6213 Project 2

Due: Wednesday, April 5, 2023, 11:59 pm

For this phase you will implement a vision based 3-D pose estimator which estimates position and orientation of the quadrotor based on AprilTags [1].

1 Vision Based Pose Estimation

1.1 Environment

The data for this phase was collected using a Nano+ quadrotor that was either held by hand or flown through a prescribed trajectory over a mat of AprilTags, each of which has a unique ID that can be found in `parameters.txt`. Figure 1 shows the layout of the AprilTag mat. The tags are arranged in a 12 x 9 grid. The top left corner of the top left tag should be used as coordinate (0, 0) with the X coordinate going down the mat and the Y coordinate going to the right. Each tag is a 0.152 m square with 0.152 m between tags with the exception of the space between columns 3 and 4, and 6 and 7, which is 0.178 m. Using this information you can compute the location of every corner of every tag in the world frame.

1.2 Calibration

The intrinsic camera calibration matrix and the transformation between the camera and the robot center are given in `parameters.txt`. Two photos (`top_view.jpg` and `side_view.jpg`) are included to visualize the camera-robot transform. The vector h contains the elements of the homography matrix stored by row $h = [h_{11}, h_{12}, \cdots]$. Once you find the homography, to guarantee the solution with positive z, you will have to scale $H = H * sign(v(9,9))$; using the sign of the last element of the last column of V. Please also notice that matlab SVD directly provides the matrix V and no its transpose. Once you find the pose from the homography, you will need to transform your camera-based pose estimate from the camera frame to the robot frame using the given constant transformation so that you can compare it against the ground truth Vicon data.

1.3 Pose Estimation

The data for each trial is provided in a `mat` file. The file contains a struct array of image data called `data`, which holds all of the data necessary to do pose estimation. The format of image data struct is as follows:

1. Time stamp (`t`) in seconds.
2. ID of every AprilTag that is observed in the image (`id`).
3. The center (`p0`) and four corners of every AprilTag in the image. The corners are given in the order bottom left (`p1`), bottom right (`p2`), top right (`p3`), and top left (`p4`), see Figure 2. The i^{th} column in each of these fields corresponds to the i^{th} tag in the ID field and the values are expressed in image coordinates.
4. Rectified image (`img`).

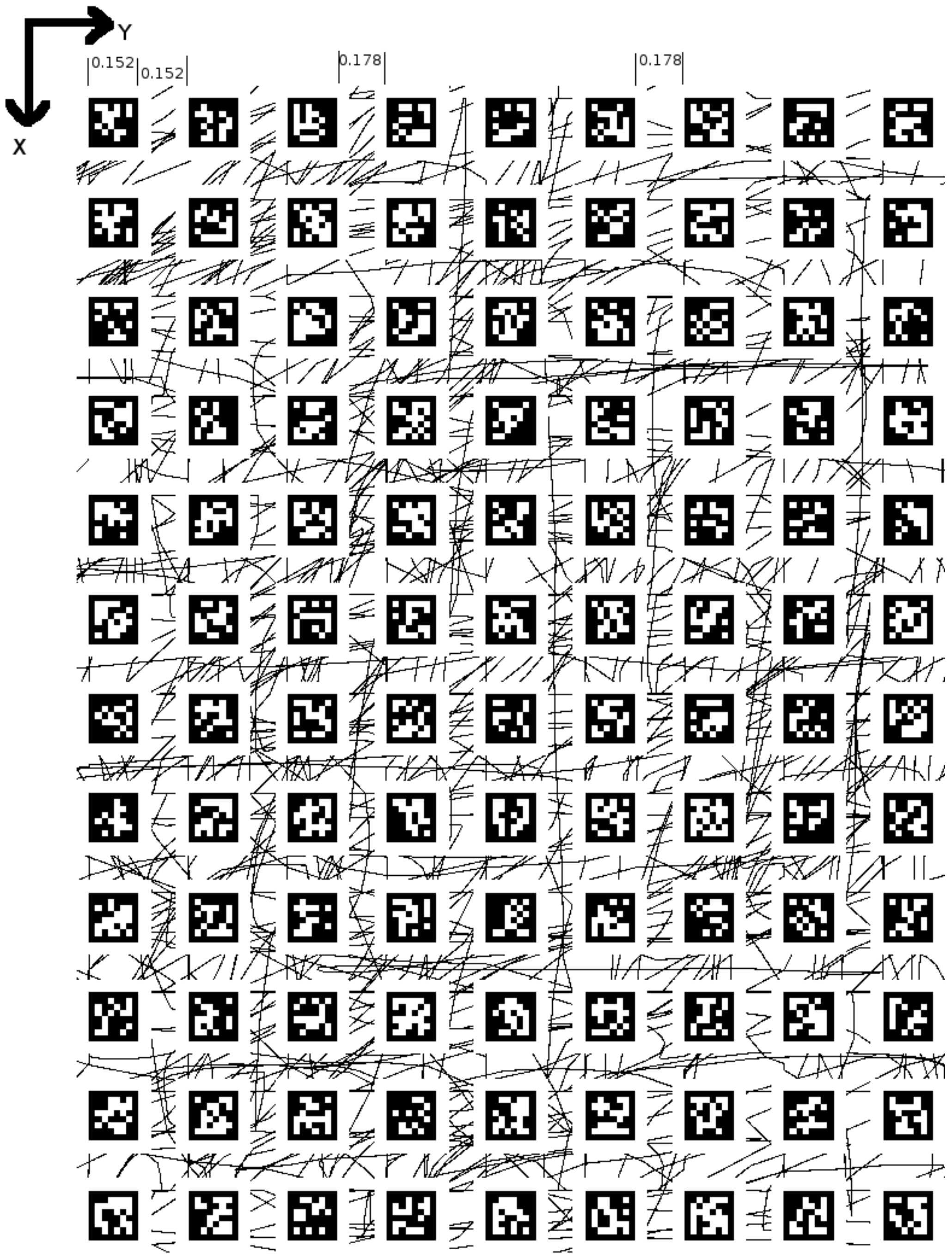

Figure 1: The AprilTag mat

The rectified images and IMU data are not necessary for this phase, but we keep them for the sake of data format consistency between this section and next one. Using the camera calibration data, the corners of the tags in the frame, and the world frame location of each tag you must compute the measured pose of the Nano+ for each packet of data.

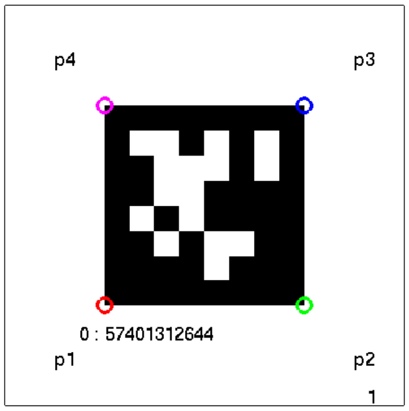

Figure 2: Corners of the AprilTag

The `mat` file also contains Vicon data taken at 100 Hz, which will serve as our ground truth measurements. The Vicon data is stored in two matrix variables, `time` and `vicon`. The `time` variable contains the timestamp while the `vicon` variable contains the Vicon data in the following format:

$$\begin{bmatrix} x & y & z & \text{roll} & \text{pitch} & \text{yaw} & v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

2 Corner Extraction and Tracking

You will first need to extract corners in each image. You may choose to use MATLAB's built in corner detectors [2, 3], external MATLAB-based computer libraries [4], find implementations on the Internet, or write your own corner detector. You are safe to assume that all detected corners will be on the ground plane. Note that the extracted corners should include not only the AprilTag [1] corners, but also corners from the random scribble.

For all corners in the image, you will then compute the sparse optical flow between two consecutive images using the KLT tracker [5]. You may again choose between [2, 3, 4], or find the tracker implementation on the Internet, or write your own implementation. This sparse optical flow, together with the time stamp of the image, will give you $[\dot{x}, \dot{y}]^T$ in the calibrated image frame. Note that the timestamps can have a bit of jitter, which adds noise to the time measurement. You can assume that the images were taken approximately at a constant rate and simply use a low-pass filter on the Δt to get better results.

2.1 Velocity Estimation

Given a corner in the calibrated image frame $[x, y]^T$ and its optical flow $[\dot{x}, \dot{y}]^T$, the following linear equation holds:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1(x, y, Z) \\ \mathbf{f}_2(x, y, Z) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{1}$$

where $[V_x, V_y, V_z, \Omega_x, \Omega_y, \Omega_z]^T$ are the linear and angular velocities to be estimated. Both the functions $\mathbf{f}_1(.)$ and $\mathbf{f}_2(.)$ return 1×6 vectors. Z is the depth of the corner. Assuming that all corners are on the floor, Z can be computed based on the estimated height and rotation of the quadrotor using the AprilTags (part of the results from Project 2, Phase 1). Note that Z does not necessarily equal the height of the quadrotor due to nonzero roll and pitch.

2.2 RANSAC-Based Outlier Rejection

It is likely that there are outliers in the optical flow computation. You will need to reject outliers using RANSAC. Three sets of constraints are required to solve the system above (1), and thus a 3-point RANSAC can be used for outlier rejection. It is recommended to recompute the velocities using (1) with all inliers.

3 Report

You need to summarize your results in a report submitted in pdf format and generated with latex or word. Please add on top of your manuscript your name and NYU ID. The report should not be more than 8 pages including plots. You will have to use the plot function that has been provided in the code to generate your results for the 2 datasets. In addition to the results, please include your approach. Do not just write equation, but try to add your logic process and explain why and how you used the equations or models you have in your code. Moreover, briefly comment your plots and compare your results to the vicon data. The plot function already overlaps your filter estimates with the ground truth provided by the vicon. Your code in each part should not take more than 1 minute to run. We will not consider submitted codes that go over this running time.

4 Grade Policy and Submission

The overall score will be 100 and will be subdivided in the following way, part 1 on pose estimation (40 points), part 2 on velocity estimation (30 points), part 2 on ransac (20 points), and report as well as code quality and readability (10 points). Do not modify any part of the code except the files estimatePose.m, getCorners.m, velocityRANSAC.m, and the dataset number to test the 2 datasets we provided. Add a flag to activate or deactivate the RANSAC part in the velocity estimation. Any other type of modification will result in 0 points. All the files, including code and report, should be submitted in an unique zip file.

References

[1] April Tags, http://april.eecs.umich.edu/wiki/index.php

[2] MATLAB Image Processing Toolbox, http://www.mathworks.com/products/image/

[3] MATLAB Computer Vision System Toolbox, http://www.mathworks.com/products/computer-vision/

[4] VLFeat, http://www.vlfeat.org/

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in Proc. of the Intl. Joint Conf. on Artificial Intelligence, Vancouver, Canada, Aug. 1981, pp. 24-28.